Estructuras de Datos 1 - ST0245
Examen Parcial 1

Departamento de Informática y Sistemas
Universidad EAFIT

1. **Teoría de la complejidad 10 %:**
   a. **(3%)**Que es el **análisis asintótico**:
      I) Método de descripción de una función cuando tiende a su valor límite.
      II) Método **informal** de analizar una función cuando tiende a su valor límite.
      III) **Único** método para comparar dos algoritmos.
      IV) Método **para contar cuantas instrucciones** ejecuta un programa en 1 segundo.

   b. **(2%)**Por qué es importante utilizar la notación O grande para comparar la complejidad de distintos algoritmos:
      I) Porque permite visualizar el máximo de instrucciones que ejecuta un programa en 1 segundo.
      II) Porque permite comparar dos o más funciones para un mismo algoritmo.
      III) Porque analiza el mejor caso de un algoritmo y así puedo determinar si mi programa es eficiente.
      IV) Porque **solo** compara dos algoritmos en el caso promedio.

   c. **(2%)**Supongamos que P(x) es una función cuya complejidad asintótica es
      O (n * m) y que A (x) es una función que está dentro de otra función H(x) cuya complejidad asintótica (de H(x)) es O (m * A (x)). Cuál de las siguientes funciones definitivamente **NO** podría ser la complejidad asintótica de A(x), sí tenemos en cuenta que P(x) > H(x).
      I) O (log n)
      II) O ($\sqrt{n}$)
      III) O (n + m)
      IV) O (1)

   d. **(3%)**Consideremos las siguientes funciones. Donde se tiene que $(\forall n \wedge \forall m)(m > n)$:
      N(n) = 2n * 3m
      M(n) = C n + 3 *n + 5*$\sqrt{n}$ + n$\sqrt{n}$
      D(n) = Cm + 2 * $n \log n$ + C
      ¿Cuál de las siguientes afirmaciones es falsa? Si hay múltiples respuestas escoja una de ellas.
      i) O(N(n) + M(n)) = O (n$\sqrt{n}$)
      ii) O(M(n) + D(n)) = O (m + $n \log n$)

iii) O(M(n) * D(n)) = O (max(M(n), D(n))

iv) O(D(n) + (M(n) * N(n)) = O(max(D(n), M(n) * N(n)) = O(M(n) * N(n)) = O (n$\sqrt{n}$* n * m) = O ($mn^2\sqrt{n}$)

2. **Complejidad Iterativa 10%**
   a. **(5%)**Dayla es un elefante muy travieso y un fanático en la solución de problemas. Un día Dayla encontró el siguiente código y de inmediato él quiso sacarle la complejidad asintótica.

```
void mistery(int n, int m)
{
  for(int i = 0; i < m; ++i)
  {
    /* P(n) es una funcion
cualquiera
    que ejecuta sus instrucciones
    en O(√n)*/
    boolean can = P(n);
    if(can)
    {
      for(int i = 1; i * i <= n; ++i)
      {
        //Hacer algo en O(1)
      }
    }
    else
    {
      for(int i = 1; i <= n; ++i)
      {
        //Hacer algo en O(1)
      }
    }
  }
}
```

Dayla sabe que la complejidad asintótica de la función P (n) es $\sqrt{n}$. Ayúdale a Dayla a sacar la complejidad asintótica para la función **mistery**.

I. La complejidad es O (m + n).
II. La complejidad es O (m * n * $\sqrt{n}$).
III. La complejidad es de O (m * n + $\sqrt{n}$)
IV. La complejidad es de O (m * n).

   b. **(5%)**Supongamos que hay una función que contiene n ciclos anidados, que ejecutan m instrucciones cada uno. Asumiendo que el resto de la función se ejecuta todo en O (1), ¿Debería ser válida la siguiente afirmación?
   T(n) = H(resto) + H(ciclos)
   O(T(n)) = O(H(resto) + H(ciclos))
   O(T(n)) = O(max(H(resto), H(ciclo))
   O(T(n)) = O(ciclos)
   O (ciclos) = O (n * m^n)

   Verdadero__ Falso__

   Nota: El resto es lo que queda en la función y ciclos es la parte de la función que contiene los bucles anidados.

3. **Complejidad recursiva 15 %**
   a. **(15%)**Oh, no, Kefo el amigo de Dayla ha sido atrapado por el malvado doctor Tenk. Tenk es un científico muy astuto y ahora quiere hacer varios experimentos con Kefo y para que Kefo no se vuele Tenk ha construido una serie de obstáculos y los ordena de menor a mayor de acuerdo a su altura. En seguida, Tenk deja a Kefo en alguno de los obstáculos. Como Kefo es muy débil y no es capaz de saltar entre obstáculos ha llamado a Dayla para que le ayude. Dayla

ha construido un programa para una máquina que valla y traiga a Kefo de su lugar de origen. La máquina simplemente salta entre obstáculo y obstáculo hasta que encuentra a Kefo y lo trae de regreso. Como hay por lo menos $10^9 + 7$ obstáculos, Dayla quiere construir un programa ***eficiente*** que traiga a Kefo. De esta manera, Kefo te entrega este código y quiere que encuentres su **complejidad asintótica** y le digas si el programa es ***eficiente***, asumiendo que el método con mayor complejidad en el programa es: ***traer_a_Kefo (kefoPos: integer): void***

```
class TraerKefo{
  private int MAXN = 10101;
  private int[] objs = new
int[MAXN];
  void traerKefo(int kefoPos){
    int where =
traer_a_Kefo(kefoPos, objs, 0,
objs.length);
    System.err.printf("trayendo a
kefo de %d\n", kefoPos);
  }
  int traer_a_Kefo(int kp, int[] o,
int l, int r){
    /*El operador >> mueve un bit
todos los bits a la derecha, en
otras palabras
    hace "(l + r) / 2" */
    int mid = (l + r) >> 1;
    if(l > r){
      return 0;
    }else if(o[mid] == kp){
      return pa;
    }else if(o[mid] > kp){
      return traer_a_Kefo(kp, o, l,
mid - 1);
    }else{
      return traer_a_Kefo(kp, o,
mid + 1, r);
    }
  }
}
```

Nota: Dayla considera un programa eficiente si su complejidad asintótica es menor O (n). Tenga en cuenta que el programa no puede tener complejidad asintótica O(n) o mayor.

4. **Teoría de Listas 5%**

   Es más óptimo buscar un elemento que no se encuentra en una lista o agregar un elemento en la posición i de una lista.

   _____
   _____

5. **Teoría de Colas 10%**
   Hay un método con dos ciclos anidados, donde cada uno ejecuta n instrucciones y en el ciclo de más adentro por cada iteración inserto un elemento en una cola, ¿Cuál es la complejidad del método asumiendo que el resto del código dentro del método se ejecuta en O (1) ?.
   i) O (n^2)
   ii) O $(n^2 + n)$
   iii) O (n^3)
   iv) O $(n^2 + \log n)$
6. **Teoría de Pilas 10%**

   Supongamos que hay un método cuya complejidad asintótica es O (n * m). Se sabe que el método solo contiene un ciclo que ejecuta m instrucciones y que una instrucción de esas ejecuta n instrucciones y las demás ejecutan O (1) instrucciones. También sabemos que la instrucción que ejecuta n instrucciones es una

pila que está haciendo una operación especifica. ¿Cuál de las siguientes operaciones está ejecutando la pila?

i. Agregar un elemento.

ii. Buscar un elemento en la pila.

iii. Eliminar un elemento de la pila.

iv. Ninguna de las anteriores.

7. **Pilas, Colas, Listas y Recursión 40%**

Dayla y Kefo fueron al bosque encantado y fueron capturados por el malvado doctor Kent, que como es costumbre intentará hacer experimentos con sus cerebros. Ahora Kent los ha subido al último piso de la torre Duck. Sin embargo, Dayla es demasiado inteligente y ha logrado salirse de una cárcel donde lo tenía Kent, pero desgraciadamente olvido como hizo para salirse de la cárcel y ahora él quiere sacar Kefo. Dayla sabe que Kefo con un poder de P unidades de energía puede reventar las cadenas de la cárcel y salirse. Pero para que Kefo tenga P unidades de energía, Kefo tiene que darle n papayas, m hamburguesas y k naranjas en su respectivo orden, ya que, si se las da en cualquier orden diferente, podría matar a Kefo. Dayla también sabe que cada uno de esos productos está en las diferentes oficinas del doctor Kent y que siempre en cada oficina hay amenos uno de ellos y no hay un producto diferente a los tres.

Como Dayla es muy astuto no quiere hacer todo ese trabajo manual y ahora se ha propuesto elaborar un robot recolector de productos. Dayla ya ha terminado por completo el robot y quiere escribir un programa para que el robot recolecte los productos. Ayúdale a Dayla a construir el programa (en Java) para que el robot recolecte los productos que necesita en la menor cantidad de pisos posible. Adicional a esto Dayla quiere que el robot siempre recoja todos los productos que hay en el piso inmediatamente inferior.

La entrada:

La primera línea de entrada consiste de 4 enteros **N, n, m, k**. **N** el número de pisos en el edificio. **n** la cantidad de papayas que necesita Kefo. **m** La cantidad de hamburguesas que necesita Kefo. **k** la cantidad de naranjas que necesita Kefo. Las siguientes N líneas consisten de 3 números (1, 2, o 3), donde 1 indica que hay una papaya, 2 indica que hay una hamburguesa y 3 indica que hay una naranja. Mire que una posible N-línea podría contener la secuencia 1 1 3 ó 3 3 2.

La salida:

La menor cantidad de pisos que necesita recorrer el robot para lograr recoger las **n** papayas, **m** hamburguesas y **k** naranjas.

Ejemplo de entrada:

4 3 3 2
1 1 1
1 2 2
2 3 3
1 2 3

Ejemplo de salida

3

Complete el siguiente código en Java.

```
import java.util.Scanner;
import java.util.Queue;
import java.util.LinkedList;
public class Solution
{
  private final int MAXN = 1000;
  private int N, K, n, m, k;
  private Queue<Integer> q1, q2, q3;
  public Solution()
  {
    q1 = new LinkedList();
    q2 = new LinkedList();
    q3 = new LinkedList();
  }
  public static void main(String[]
args)
  {
    Solution ans = new Solution();
    ans.go();
  }
  void go()
  {
    Scanner sc = new
Scanner(System.in);
    N = sc.nextInt();
    n = sc.nextInt();
    m = sc.nextInt();
    k = sc.nextInt();
    int result = 0;
    boolean can = true;
    for(int i = 1; i <= N; ++i){
      int a, b, c;
      a = sc.nextInt();
      b = sc.nextInt();
      c = sc.nextInt();
      evalRec(new int[]{a, b, c}, 2);
      if(solve(i) && can)
      {
        result = i;
        can = _______;(7%)
      }
    }

System.out.println(___________);(7%
)
  }
  void evalRec(int [] a, int i){
    if(i < 0){
      return;
    }else if(a[i] == 1){
      q1.add(0);
      evalRec(a, i - 1);
    }else if(a[i] == 2){
      q2.add(0);
      ___________;(7%)
    }else{
      q3.add(0);
      ___________;(7%)
    }
  }
  boolean solve(int i)
  {
    if(q1.size() >= n && q2.size() >=
m  && ________)(7%)
    {
      return _____________;(5%)
    }
    return false;
  }
}
```